Examiner's Amendment and Statement of Reasons for Allowance

1. This action is responsive to Applicant's amendment filed August 14, 2008.

2. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on August 14, 2008 has been entered.

Examiner's Amendment

3. An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

Authorization for this examiner's amendment was given in a telephone interview with Mr. Ralph F. Hoppin, Registration Number 38,494, on October 07, 2008 for including all the distinct features of the invention, and put the claims in condition for allowance.

The application has been amended as follows:

1 (currently amended) A method for adding functionality in order to access information, comprising:

accessing existing object code of a first application, said existing object code includes a first method, said first method produces a result when said first method is executed, said existing object code is not configured to provide access to said result by an additional method for which such access is desired;

editing said existing object code to provide modified object code which provides access to said result by said additional method when said first method is executed; and

executing said modified object code;

said editing comprises modifying said existing object code for:

storing said result, said result is from an operand stack:

preparing said operand stack for an invocation of said additional method;

invoking said additional method, including providing said result to said additional method; and

resetting said operand stack with respect to said result to a state existing prior to said step of storing said result.

4. (cancelled)

5. (cancelled)

6. (cancelled)

7. (currently amended) A method according to claim 1 [[4]], wherein said step of editing includes:

adding code that jumps to a subroutine representing a Finally block after invoking said additional method; and

adding code that is to be executed after returning from said subroutine.

8. (cancelled)

12. (cancelled)

13. (currently amended) A method according to claim 1 [[12]], wherein:
said result includes a data item to be returned by said first method.

15. (currently amended) A method according to claim 1 [[12]], wherein:
said result includes a reference to an exception.

17. (currently amended) A method according to claim 1 [[12]], further comprising:
performing said ~~second~~ additional method in response to said step of invoking.

18. (currently amended) A method according to claim 1 [[12]], further comprising:
performing one or more instructions of said first method prior to said step of storing said result, said step of performing one or more instructions includes generating said result.

19. (currently amended) A method according to claim 1 [[12]], in, further comprising:
returning said result subsequent to said step of resetting;

jumping to a subroutine representing a Finally block after invoking said ~~second~~ additional method and prior to returning said result; and
returning from said subroutine prior to returning said result.

20. (cancelled)

21. (currently amended) A method according to claim 1 [[12]], wherein ~~further comprising:~~
~~modifying byte code for said first method, said byte code is not configured to provide said result to said second method when said second method is invoked, said modifying includes configuring~~ said modified object code comprises byte code which performs ~~to perform~~ said steps of storing, preparing, invoking and resetting.

22. (currently amended) A method according to claim 21, wherein said step of editing ~~modifying~~ includes:
adding start byte code;
adjusting byte code indices;
adding exit byte code; and
modifying an exception table for said first method.

24. (currently amended) A method according to claim 21, wherein said step of ~~modifying~~-editing includes:
adding Try-Finally functionality which is not included in ~~said~~ byte code for said first method which is modified.

25. (currently amended) A method according to claim 1 [[12]], further comprising: performing said ~~second~~ additional method, including accessing said result.

26. (currently amended) A method according to claim 1 [[12]], further comprising: performing said ~~second~~ additional method, including storing said result for use outside of a thread that includes said first method.

27. (currently amended) A method according to claim1 [[12]], further comprising:

performing said ~~second~~ additional method in response to said step of invoking, said ~~second~~ additional method stores said result;

performing one or more instructions of said first method prior to said step of storing said result, said step of performing one or more instructions includes generating said result~~, said first method is a JAVA method~~; and

modifying byte code for said first method to perform said steps of storing, preparing, invoking and resetting.

28. (currently amended) A method according to claim 27, further comprising:

returning said result;

jumping to a subroutine representing a Finally block after invoking said ~~second~~ additional method and prior to returning said result; and

returning from said subroutine prior to returning said result.

29. (currently amended) One or more processor readable storage devices having processor readable code embodied on said processor readable storage devices, said

processor readable code for programming one or more processors to perform a method comprising:

accessing existing object code, said existing object code includes a first method, said first method produces a result when said first method is executed, said existing object code is not configured to provide access to said result by an additional method for which such access is desired; and

modifying said existing object code to provide modified object code that, when executed, provides said result to said additional method;

said step of modifying includes:

adding code that stores said result for said first method from an operand stack:

adding code that prepares said operand stack for an invocation of said additional method;

adding code that invokes said additional method, including providing said result to said additional method; and

adding code that resets said operand stack with respect to said result to a state existing prior to storing said result.

32. (cancelled)

33. (cancelled)

36, (currently amended) An apparatus that adds functionality in order to access information, comprising:

a communication interface;

a processor readable storage device; and

one or more processors in communication with said processor readable storage device and said communication interface, said one or more processors perform a method comprising:

access existing object code, said existing object code includes a first method, said first method produces a result when said first method is executed, said existing object code is not configured to provide access to said result by an additional method for which such access is desired; and

modifying said existing object code to provide modified object code that, when executed, provides said result to said additional method;

said step of modifying includes:

adding code that stores said result for said first method from an operand stack;

adding code that prepares said operand stack for an invocation of said additional method;

adding code that invokes said additional method including providing said result to said additional method; and

adding code that resets said operand stack with respect to said result to a state existing prior to storing said result.

38. (cancelled)

39. (currently amended) An apparatus according to claim 36, wherein said step of modifying includes:

adding start ~~JAVA~~-byte code;

adjusting ~~JAVA~~-byte code indices;

adding exit ~~JAVA~~ byte code; and

modifying an exception table for said first method.

40. (currently amended) An apparatus according to claim 39, wherein said step of adding exit ~~JAVA~~ byte code includes:

adding byte code to report said result and jump to a subroutine representing a Finally block;

adding byte code to report an exception and jump to said subroutine representing said Finally block; and

adding byte code for said subroutine representing said Finally block.

41. (currently amended) An apparatus that adds functionality to existing code in order to access information, comprising:

a communication interface;

a processor readable storage device; and

one or more processors in communication with said processor readable storage device and said communication interface, said one or more processors perform a method for modifying byte code for a first method, said byte code is not configured to provide a result to a second method when said second method is invoked, said method comprising:

storing a result for said [[a]] first method from an operand stack,

preparing said operand stack for an invocation of said [[a]] second method,

invoking said second method, including providing said result to said second method, and

resetting said operand stack with respect to said result to a state existing prior to said step of storing said result.

43. (cancelled)

44. (cancelled)

45. (cancelled)

46. (currently amended) An apparatus according to claim 41 [[45]], wherein said step of modifying includes the steps of:

adding start byte code;

adjusting byte code indices;

adding exit byte code; and

modifying an exception table for said first method.

-- The End --

Examiner's Statement of Reason(s) for Allowance

4. Claims 1-3, 7, 9-11, 13-19, 21-31, 34-37, 39-42, 46-52 are allowed.

5. The following is an examiner's statement of reasons for allowance:

The prior arts of record: **Nilsson**, teaches using in-code context data for exception handling is incorporated into a special call instruction which is recognized by the processor. The information is skipped at the time of the function call and read at the time of the stack unwinding. **Kukol**, teaches a system having a compiler that allows programmers and software developers to more efficiently develop compiled applications with runtime exception handling support is described. The compiler implements methods for handling of exceptions, which may occur during runtime execution of the program. In an exemplary embodiment, the system of the present invention registers exception handling information (e.g., an Exception Registration Record) with the underlying operating system, during execution of prolog code for each function (or other discrete section of code). **Van Dyke** et al., teaches computer has a multi-stage execution pipeline and an instruction decoder. The instruction decoder is designed (a) to decode instructions of a complex instruction set for execution by the pipeline, the instruction set being architecturally exposed for execution by user-state programs stored in a main memory of the computer, the instruction set having variable-length instructions and many instructions having multiple side-effects and a potential to raise multiple exceptions, (b) for at least some instructions of the complex instruction set, to issue two or more instructions in a second internal form into the execution pipeline; (c) to generate information descriptive of instructions to be executed by

the pipeline, and to store the information into non-pipelined processor registers of the computer; and (d) to determine whether instructions will complete in the pipeline, and to abstain from writing descriptive information into the processor registers for instructions following an instruction determined not to complete.

Yellin et al., teaches a program interpreter for computer programs written in a bytecode language, which uses a restricted set of data type specific bytecodes. The interpreter, prior to executing any bytecode program, executes a bytecode program verifier procedure that verifies the integrity of a specified program by identifying any bytecode instruction that would process data of the wrong type for such a bytecode and any bytecode instruction sequences in the specified program that would cause underflow or overflow of the operand stack. **Berry** et al., teaches a user may specify a vector of metrics to be used while profiling a program. The vector of metrics may optionally be thread-relative. In response to a notification of an occurrence of the current event, a thread-relative elapsed metric is computed by: determining a current thread; retrieving a stored reference metric for the preceding event of the current thread; obtaining a current reference metric; and computing the thread-relative elapsed metric as a difference between the current reference metric and the stored reference metric. **Chow** et al., teaches method for translating exception handling semantics of a bytecode class file within a computer system is disclosed. An empty bytecode information array is first established. Pertinent information is then obtained by examining bytecodes within a bytecode stream, and such information are inserted into the bytecode information array. An exception framelist, which includes an exception handling routine, is subsequently obtained from a class file for the bytecode stream. The entries within the bytecode information array corresponding to a

starting location and an ending location of the exception framelist are marked. Finally, a high-level code sequence is generated utilizing the bytecode information array. **Crank** et al., teaches a method and apparatus for performing runtime checking during program execution in a compiled environment using the full ANSI-C programming language. The present invention detects a number of errors during runtime that cannot be found by a compiler at the precise moment that a respective C language restriction is violated. **Morshed** et al., teaches techniques for gathering execution information about an application, such as a distributed application, are described. Key communication points in cross execution context calls, such as remote procedure calls, are determined and control is transferred to interception routines to insert and extract execution information. Outgoing remote procedure calls are intercepted on a client that inserts call origin information into the request sent to a server system. The server system intercepts and extracts the call origin information and additionally inserts other information in a response sent to the client system upon completion of a remote procedure call. **Kramskoy** et al., teaches a dynamic compiler and method of compiling code to generate a dominate path and handle exceptions. The dynamic compiler includes an execution history recorder that is configured to record the number of times a fragment of code is interpreted. When the code is interpreted a threshold number of times, the code is queued for compilation. **Yates** et al., teaches a computer concurrently executes a first operating system coded in a RISC instruction set and a second operating system coded in a CISC instruction set. When an exception is raised while executing a program coded in the RISC instruction set, an execution thread may be initiated under the CISC operating

system. **Draine** et al., teaches an error/exception helper may provide tailored help when an error such as an exception is generated. A source program editor interface may be displayed and/or focus given to the program editor interface. An error/exception bubble or tool tip may be displayed, which, in one embodiment of the invention, points to the line of code that generated the exception. The error/exception bubble may include a link to a help topic or the actual help text may be displayed within the bubble. New arts made of record: US Patent No. 6,904,517 by **Nevill** et al., teaches a data processing apparatus and method for saving return state. The data processing apparatus comprises a processing unit for executing data processing instructions, the processing unit having a plurality of modes of operation, with each mode of operation having a corresponding stack for storing data associated with that mode. The processing unit is responsive to a return state data processing instruction to write return state data of the processing unit from its current mode of operation to a stack corresponding to a different mode of operation to the current mode of operation. US Patent No. 5193180 by **Hastings**, teaches an object code expansion program inserts new instructions and data between preexisting instructions and data of an object code file; offsets are modified to reflect new positions of the preexisting instructions and data. For each item of preexisting object code (instructions or data), the following steps are performed: making a new code block comprising any desired new instructions and the item, and storing it as new object code; tracking the location of the item and the new code block within the new object code; and tracking items that contain inter-item offsets. Then, each inter-item offset is updated using the new location of the item or new code block, as required. US Patent No. 5,423,042 by **Jalili** et al., teaches a computer server program is disclosed that can execute object code

provided by one or more clients even though the server was not previously programmed to execute specific code. The server executes the client code without re-compiling or re-linking. Clients provide the server with executable object code along with information about the code that allows the server to register the code object. US Patent No. 6,901,588 by **Krapf** et al., teaches a method and system for representing and implementing a concept between two functional domains (e.g., programming languages) by using a proxy component in a first domain to wrap a component of a second domain, where the proxy component has a semantic usability in the first domain closely corresponding to the semantic usability of the underlying component from the second domain. Further, provided is a method and system for automatically generating such a proxy component. US Patent No. 6,067,641 by **McInerney** et al., teaches a human-oriented object programming system (HOOPS) and its debugger provide an interactive and dynamic modeling system to assist in the incremental generation of symbolic information of computer programs that facilitates the development of complex computer programs such as operating systems and large applications with graphic user interfaces (GUIs). A program is modeled as a collection of units called components. However, none of them, taken alone or in combination, teaches the features in such a manner as recited in independent claims 1, 29, 36, and 41.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

6. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Chih-Ching Chow whose telephone number is 571-272-3693. The examiner can normally be reached on 8:00am - 4:30pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Zhen can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Chih-Ching Chow/

Examiner, Art Unit 2191

10/08/2008

/Wei Y Zhen/

Supervisory Patent Examiner, Art Unit 2191